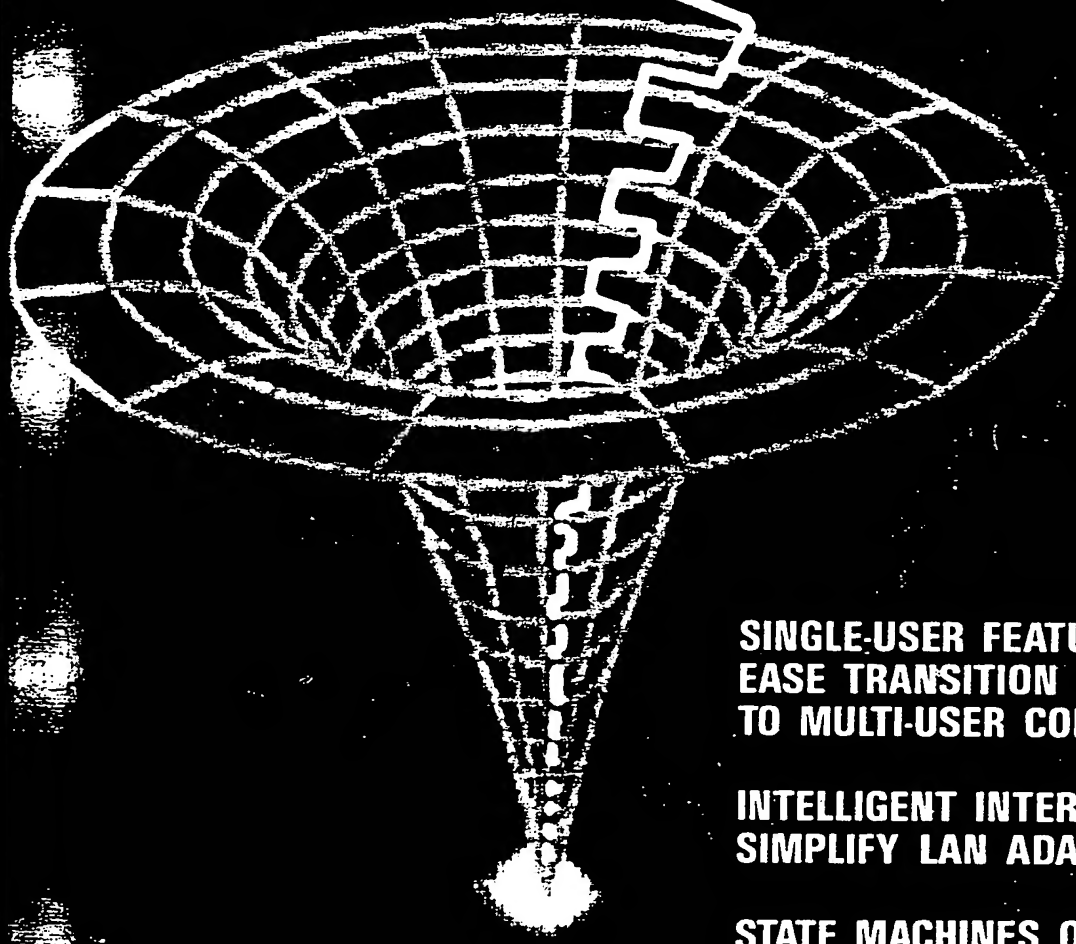
PennWell Publication

COMPUTER DESIGN

MAY 1985

DISK AND TAPE STORAGE



SINGLE-USER FEATURES EASE TRANSITION TO MULTI-USER COMPUTING

INTELLIGENT INTERFACE ICs SIMPLIFY LAN ADAPTERS

STATE MACHINES OFFER FAST PATTERN MATCHING

CONTROLLERS WRING PEAK PERFORMANCE OUT OF DISK DRIVES

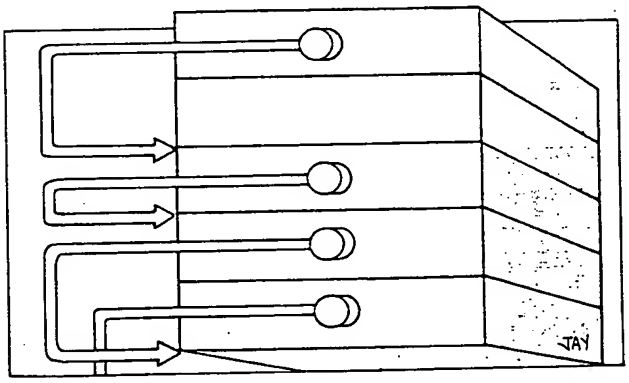
Elevator sorting, disk cache, and random data storage techniques, implemented with the help of advanced disk controllers, minimize performance limitations due to head positioning time.

by Mark S. Young

Today's disk subsystems provide large amounts of memory and fast access with minimal cost, at levels designers could only dream about a few years ago. Continual increases in track and linear bit density—and, as a result, in transfer speeds—combine with decreases in access times to improve performance of disk products. Disk controllers and operating systems have also enjoyed steady gains that increase speed. Improvements in processor speeds, however, have led system integrators to demand still higher throughputs.

Yet, several design techniques can still yield better overall data throughput. Typical disk drives require between 90 and 30 ms for a seek operation. Since the read/write head positioning, or seek, consumes the most time in a disk I/O operation, controller and operating system designers have focused their efforts there.

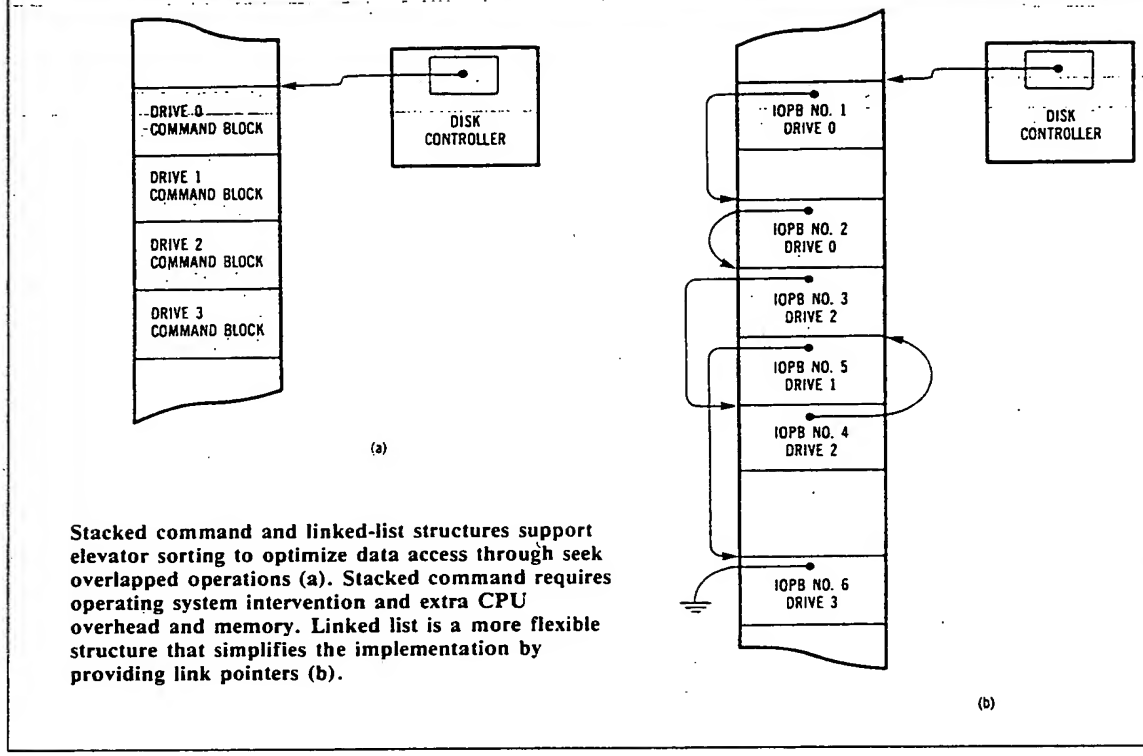
In a multidrive system, controller design can change the sequential series of disk access operations into parallel ones. A typical 5¼-in. Winchester, using the ST506 interface, can accept head movement pulses about once every 10 ms (1 pulse means to move the read/write heads from one track to the next). A read/write head requires up to 3 ms to move from track to track, depending on how far it has to move. Most small Winchester disks do not require selection while read/write head positioning is taking place. After the controller issues a complete head movement command, therefore, it can turn its attention to another drive while the positioning operation occurs.



Optimizing seek operations

To execute disk operations on different drives more or less in parallel, the controller looks at multiple disk commands from the system and isolates the commands to the different drives. After determining the operations that are required, the controller issues necessary commands to the drive. It

Mark S. Young is a product planning engineer at Advanced Micro Devices (Sunnyvale, Calif). He holds a BA in computer science from the University of California at Berkeley.



Stacked command and linked-list structures support elevator sorting to optimize data access through seek overlapped operations (a). Stacked command requires operating system intervention and extra CPU overhead and memory. Linked list is a more flexible structure that simplifies the implementation by providing link pointers (b).

then processes another drive's seek operations while waiting for the current drive's command to be completed. Average access time to all the disks is substantially improved. In a four-drive system, this could be a gain of as much as 400 percent.

The success of seek-overlap operations, as these parallel seek operations are called, depends on random disk file request. If the computer's operating system tends to access files on a single disk consistently (in a multidrive system), then seek-overlap has little advantage. Maximum benefit is gained when files are stored on a randomly chosen disk. In some Unix operating systems, for example, the disk file manager randomly selects a disk on which to store a newly created file. This ensures the disk controller will receive I/O access requests that are distributed among the various drives.

Since files are not usually accessed in a truly random manner, the randomizing technique may require adjustments. By keeping track of all file accesses over a certain period, the operating system can redistribute the files among different disks. Although extra disk accesses and additional computation overhead are required, access time improvement can be significant in a heavily loaded system.

Another technique, called elevator accessing, can also reduce the average seek time. This technique sorts track access operations into two groups: seeks that move steadily toward the disk spindle (toward the inner tracks) and those that move steadily away from the spindle (towards the outer tracks). In this grouping, each successive access is in the same direction. The read/write heads move first in one direction and then the other, accessing requested data as they go. This eliminates back and forth motions of the head. The sorting function generally is handled by either the disk file controller or the computer operating system.

A typical series of disk seek requests, if executed as received, might require a total seek distance of some 556 tracks. Sorting the incoming requests numerically could reduce this to about 283 tracks—a 50 percent reduction in seek distance and time.

File dependencies, however, mean full performance benefits cannot always be gained from optimal elevator sorting. If one request requires a write operation on a track and a later operation requires a read, the elevator sort must maintain the correct sequence of events. Such file dependencies can be controlled by designating which accesses must occur first in multiple data requests from the same file. One method is to time stamp disk access requests and always give priority to the request with the earliest time.

The elevator sorting algorithm, like seek-overlap operations, will only work if the controller can queue

commands long before they will be needed. The stacked command structure is a commonly used technique that supports these functions. An area in system memory is used to store disk I/O commands, usually one command per drive. The controller can easily initiate a primitive seek-overlap algorithm by allowing up to four disks to have commands pending. Elevator sorting must be performed in the file manager or a separate disk I/O processor. To guarantee continuous use of the controller, the file manager must also update the command structure whenever a disk is free. A disadvantage of the stacked command structure is that operating system intervention is required each time a disk I/O command is completed. Thus the controller would introduce extra CPU overhead.

The linked list command chain offers a more flexible command structure. A disk command chain, created using disk command blocks (I/O parameter blocks), is placed either in consecutive memory locations or is strung together as a linked-list data structure. Although this structure imposes some extra overhead, such as pointers to commands in the list, it maximizes system flexibility when integrating the disk controller with the operating system.

Zero-interleaved data transfers improve data I/O operations, as well as speed disk access times.

The operating system creates the command structure and the disk controller executes it. The linked-list structure allows IOPBs to be placed in the command chain well before execution without imposing size limitations. Since a queue of disk I/O commands is available to the disk controller, it simplifies the implementation of a seek-overlap function in the disk controller.

Moreover, it allows the disk controller to start searching the command list for the command that starts the next seek as soon as one I/O operation finishes. This keeps the drive constantly busy without system intervention. The linked-list structure also speeds execution of the elevator access algorithm. Because the operating system can create the commands and then sort the seeks without moving commands around in system memory, only the link pointers have to be altered.

Disk I/O operations can also be improved by continuously transfering data to and from a track. This method, called zero-interleaved data transfers, sequentially accesses physical sectors on the disk. Many controller designs require sectors on a track to be interleaved. Interleaving calls for specific amounts of physical space between logically adjacent track sectors. It also provides time between processing of the data to (or from) the disk and the arrival of the next desired sector on the track.

Data buffering and high speed control capability can eliminate the need for interleaving. One method uses first in, first out buffers to allow zero-interleaved operations. FIFOs, however, are limited in size and allow underflow/overflow. A number of simple, low performance system FIFOs in various sizes (8, 32, or 128 bytes) are used in VLSI controllers to minimize hardware costs, while still allowing zero-interleaved transfers.

Alternate data transfer methods

Another method uses a pair of sector or toggle buffers. This allows disk data to fill or empty one buffer while the system empties or fills the other. Continuous data transfer is provided by switching buffers at the correct time. System memory can be completely decoupled from the disk drives, since all disk data from one transfer is contained in one of two buffers. Toggle buffers also prevent data underflows/overflows and allow on-controller correction of data errors.

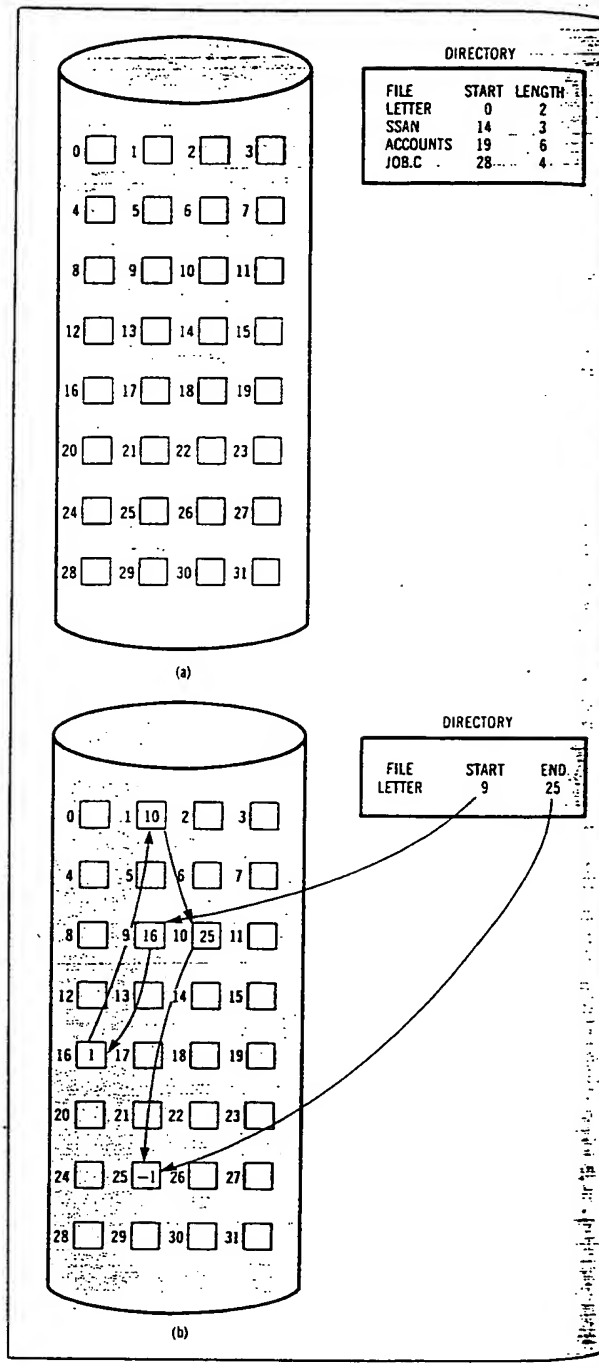
A dual-ported RAM can also implement the buffering needed to perform zero-interleaved operation. These RAMs are expensive, and require close coordination of timing control and addressing. A track buffer can hold all data from a single disk track. This RAM, however, should be designed as a variation of the dual-ported RAM to ensure continuous throughput between the disk and the system.

Whatever method is used, the zero-interleaved operation significantly improves disk access times. A typical Winchester disk rotates at 3600 rpm, requiring about 16.6 ms per revolution. With a controller interleaving of degree five on each track (logically adjacent sectors are physically separated by five sectors), at least six revolutions (or 100 ms) are required to read or write data to an entire track. This actual data transfer time, when added to the time required to move the read/write heads to a track (80 to 90 ms), results in a total I/O access time that is twice that using zero interleaving.

A variation of zero-interleaved transfers, a technique called "nonsequential" sector access, further boosts overall performance. Data is stored on the disk tracks in blocks called sectors. These sectors are logically accessed in a sequential manner for all multisector operations (ie, a controller first accesses the sector numbered A, then A + 1, A + 2, etc). A normal controller logically accesses several consecutive sectors on a track, by searching for the first desired logical sector and reading or writing data from (and to) subsequent sequential sectors.

Statistically, however, the controller will miss the desired first sector half of the time. This means that about half the amount of time a rotation takes must be added to the "access" time of every new track operation. This delay, about 8 ms on 5 1/4-in. Winchesters, is called "rotational latency" time. On fast disk drives (with average access time of 30 ms), rotational latency can increase the average access time of the disk by about 15 percent (ie, 30 ms for seeking, 16 ms for read/write, and 8 ms for rotational latency).

The nonsequential sector access method minimizes rotational latency by accessing the requested sectors in whatever order they are found. As soon as any desired sector (A, A + 1, A + 2, etc) is located, the disk controller performs the necessary I/O operation on that sector. Thus, the maximum time necessary to access any amount of data is no more than that required for one rotation of the disk plus one sector. The nonsequential access method is implemented by using a track buffer to hold the contents of the track during I/O operations. The controller, however, requires extra hardware for the logic necessary to recognize when a desired sector passes under the read/write heads.



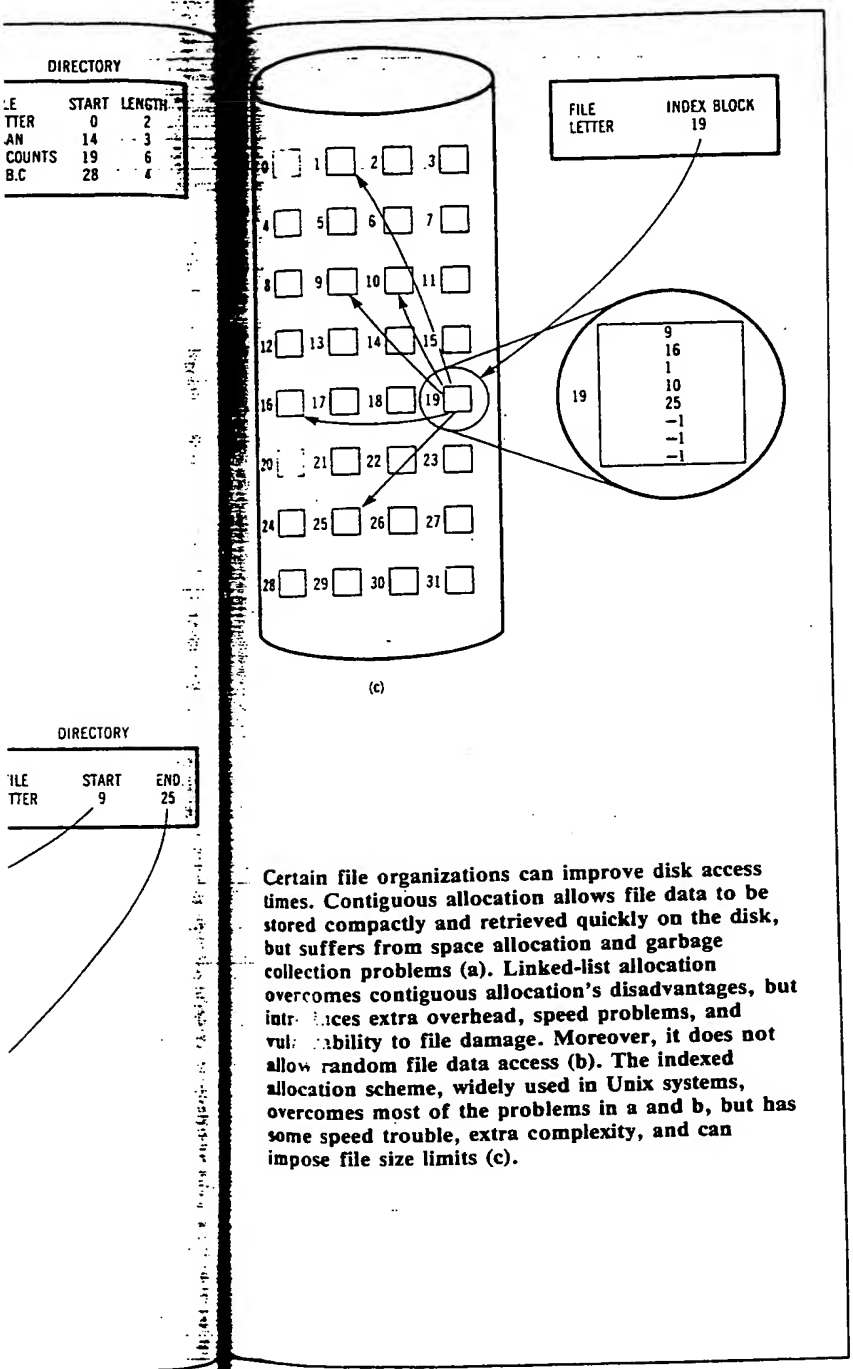
Boosting disk performance

A disk data cache provides an increasingly common method of boosting disk performance. Disk cache functions are much like those of caches used for CPUs. As data is read or written from or to the disks, copies are stored in the cache. If the operating system later requests the same data, the data can be taken from cache instead of from disk. Disk caching boosts disk I/O performance from two to nine times because head positioning delays are not involved. Since cache is not disk dependent, slow, low cost disks can be used with a cache controller to give performance approaching that of expensive, high performance drives.

Disk caching can be done either by using the operating system, or by building a dedicated cache controller. Having the operating system handle cache requires cache management software and reduces system memory space (since it must be allocated to the cache buffers). The other popular method of implementing disk caching is through dedicated cache controller hardware. This involves adding RAM, a cache manager (usually a microprocessor and ROM), software, some form of interface to the operating system, and the disk controller.

With this method, whenever a disk I/O request is initiated by the operating system, it is sent to the cache controller instead of to the disk controller. If the requested data is in the cache, it is passed to the system without disk controller intervention. If the data is not in the cache, the cache controller passes the command to the disk controller. When the data is returned from the disk, the cache controller makes a copy of the data and puts it in cache as it is sent to the system.

Organization of files on a disk can also affect access time. Normally, all read/write heads on a head assembly are positioned on the same track



Certain file organizations can improve disk access times. Contiguous allocation allows file data to be stored compactly and retrieved quickly on the disk, but suffers from space allocation and garbage collection problems (a). Linked-list allocation overcomes contiguous allocation's disadvantages, but intr[illegible]ces extra overhead, speed problems, and vul[illegible]ability to file damage. Moreover, it does not allow random file data access (b). The indexed allocation scheme, widely used in Unix systems, overcomes most of the problems in a and b, but has some speed trouble, extra complexity, and can impose file size limits (c).

ller. When the da[illegible] he controller mak[illegible] cache as it is s[illegible] can also affect [illegible] d/write heads on [illegible] on the same trac[illegible]

although they are on different surfaces. Operating systems can allocate multiple tracks for file storage on either the same or multiple surfaces. A preferred strategy is to switch read/write heads until a particular track on all surfaces is used before moving the read/write heads to another track. This is the fastest method, since the time required to switch from one read/write head to another usually requires only microseconds, while changing tracks requires several milliseconds.

A recurring problem with many disk-based operating systems is file fragmentation, which occurs when files are updated (ie, increased or decreased in size). One reason for the popularity of the Berkeley version of Unix (4.2 BSD) is its use of an improved file management system to increase file access speed and reduce fragmentation.

Files are stored on disks using either linear (continuous), linked-list allocation, or indexed allocation. The linear method stores files in single, contiguous blocks on the disk, minimizing the time needed to read the files. To increase or decrease file size, data must be moved to another area on the disk that is equal to or greater than the new file. Because files are allocated randomly, free space is broken into a large number of pieces. Frequent compaction is required to maximize the disk's storage capacity, usually at a great cost in disk processing time.

The linked-list file structure incorporates a pointer in each sector to indicate the next sector belonging to the file. Fragmentation is avoided because every sector on the disk can be used in disk files. Despite the advantages of using complete disk space and the flexibility in allocating space, the linked structure requires extra overhead in each sector for the pointers. The structure is also more vulnerable to damage; if the link in one sector is damaged, the rest of the file is lost. The operating system's inability to make direct accesses into the file is also a shortcoming.

The indexed allocation method uses a pointer table built into the beginning of the sector to define all sectors in the file. This method, like the linked list, minimizes fragmentation on the disk because all sectors can be used and direct access is allowed. The space required for the file pointer table, however, is usually fixed at file creation and must serve for the life of the file. Since estimating the table size is usually difficult, overestimation is the rule, costing storage space. The space required for the table frequently exceeds that used for the linked-list method. And unless the index tables are kept in the controller, the elevator sort algorithm cannot be properly exploited.

Programmable features in modern disk controllers combine the best features of all three file storage methods while minimizing their disadvantages. Keeping the different file techniques distinct from one another presents the biggest problem when methods are combined. An advanced controller, such as the Am9580, provides ability to flag individual sectors, allowing the sectors to determine how they are being used. For example, a combination of the linear method with the linked-list file allows allocation of a single block of memory when the file is created.

But it still allows sectors to be added without resulting in fragmentation or excessive file movement.

Ondisk file caching also boosts overall disk/system throughput. This method stores frequently used files in the most rapidly accessible part of the disk. Normally, files are stored on the disk randomly (based on such things as space requirements and partitioning of the disk data space).

Second-generation VLSI disk controllers

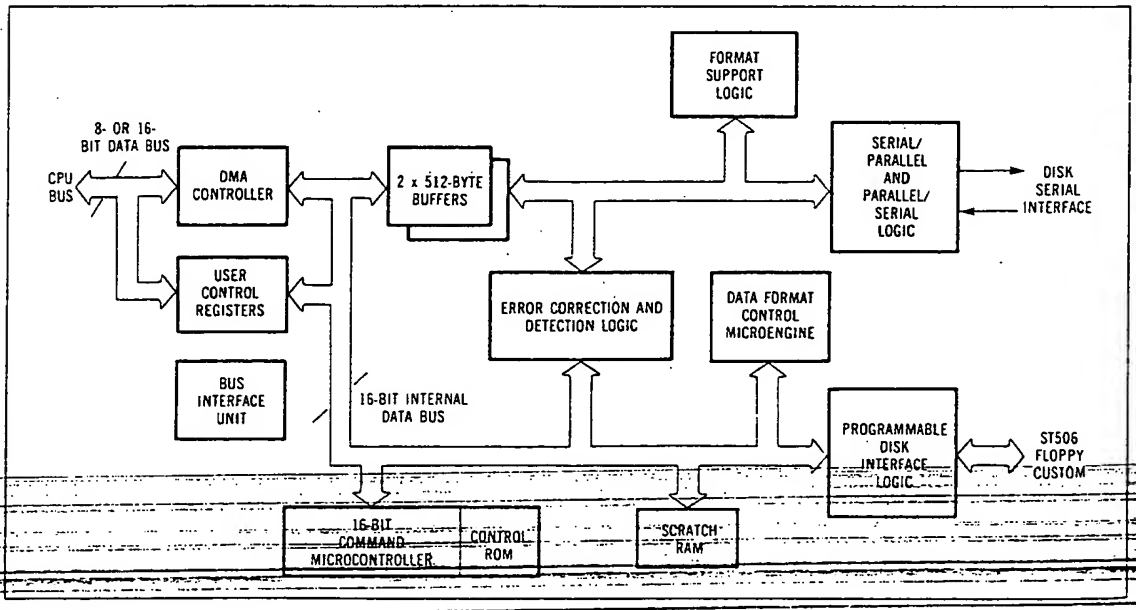
A primary requirement of second generation VLSI disk controllers is maximum disk and system performance. The Am9580 hard disk controller's dual-buffer architecture with integrated DMA controller boosts disk system performance in two ways. First, it fully decouples the disk serial interface (up to 16 Mbits/s) from the system bus. This allows the system to operate without being tied into the peculiarities of the disk timings. Second, it provides efficient data transfers and cuts software overhead. The DMA controller can transfer data at rates up to 5 Mbytes/s and supports 8- or 16-bit interfaces. Programmable bus throttling regulates bus activity of the hard disk controller on the system bus.

Another facet of second generation disk controllers is the large amount of integrated software. The Am9580 has two different microengines operating in parallel and executing specialized disk data control algorithms. One of the microengines, the data format controller, handles all the serial transfer of data to and from the disk.

In addition to performing track formatting and sector reads and writes, it handles special data recovery algorithms, and floppy or Winchester disk formats. The second microcontroller, the command sequencer, is responsible for interpreting the 16 different disk commands, coordinating DMA activity, and handling the disk control interface. Command, status, and data transfers are handled by the controller with minimal CPU intervention. The hard disk controller uses a linked-list command structure. Command blocks (called I/O parameter blocks) are set up as a linked list in system memory. The disk controller automatically fetches and executes the commands without additional CPU overhead. Data transfers between the disk and system are handled by the onboard DMA controller. An additional control structure, called data map, allows the Am9580 to break up or combine data scattered throughout system memory and collect it into contiguous blocks.

Status result information is stored in a specially designated area of system memory called the status result area. Errors resulting from abnormal disk behavior are reported to the system in the status result area along with an identification code that describes which command caused the error. Users can program how the controller will handle errors and when it will abort a command.

Finally, the Am9580 supports several different disk drive control interfaces. The industry standard ST506 Winchester interface is fully implemented. For floppy drives, a floppy-like control interface is supported. To accommodate custom disk interfaces, a special programmable option allows different portions of the disk control interface to be selectively disabled. This allows the user to implement (externally) any desired control interface.



ptimal disk access is achieved, however, by centering read/write heads around the middle tracks on the platter because the average distance to any location (assuming random disk I/O requests) is minimized. To ensure this happens, the disk file manager must keep track of how frequently different files on the disk are accessed. Then the file manager must rearrange the files so those most frequently accessed are located on the center tracks. In paged virtual memory systems, spare data blocks allocated on the outer tracks should be used to accommodate data that is constantly being swapped in and out of system memory.

Ondisk caching can be put into the operating system's normal disk I/O routines without degrading system throughput. During medium and high system load periods, the operating system should only statistically track disk I/O. Periods of low use (about 5 percent or less) can then be used by the disk sorting routines to move files around (based on current statistics) and to clean up fragmented files and put them into linear blocks. These routines would move least recently used files to outer or inner tracks, and migrate more frequently used files to center tracks. Special options would allow the operating system designer to force certain files to middle tracks or prevent the ondisk caching routines from moving them. Since many systems have long periods of low loading, daily updates of the ondisk cache would not degrade users' response time. Thus, overall throughput of the disk drives would increase without any system overhead that is visible to the user.

Disk controllers employing these techniques are not simple systems based on one or two VLSI chips. Instead, the disk controller is built up around powerful disk controller ICs, such as the Am9580 controller and the Am9581 disk data separator, and supported by a microprocessor, random logic, RAM (large amounts for caching), and ROM to contain all the necessary software algorithms. Although single-user, single-tasking microcomputers do use these techniques, newer machines and Unix-based systems require very high performance disk systems to perform adequately in most multi-user, multitasking environments.

Please rate the value of this article to you by circling the appropriate number in the "Editorial Score Box" on the Inquiry Card.

High 707 *Average 708* *Low 709*